

Symbolic Computation in Electromagnetic Modeling

Larry A. Lambe

llambe@mssrc.com

Richard Luczak

luczakr@asc.hpc.mil

John W. Nehrbass

nehrbass@osc.edu

June 21, 2001

- A New Finite difference method for the Helmholtz equation was given in [Nehrbass, 1996]

- Optimally adjust the standard weights

- 1D:

$$u'' \approx \frac{u(x+h) - w u(x) + u(x-h)}{h^2}$$

standard: $w = 2$; adjusted: $w = 2 \cos(\kappa h) + (\kappa h)^2 = 2 + O(h^4)$

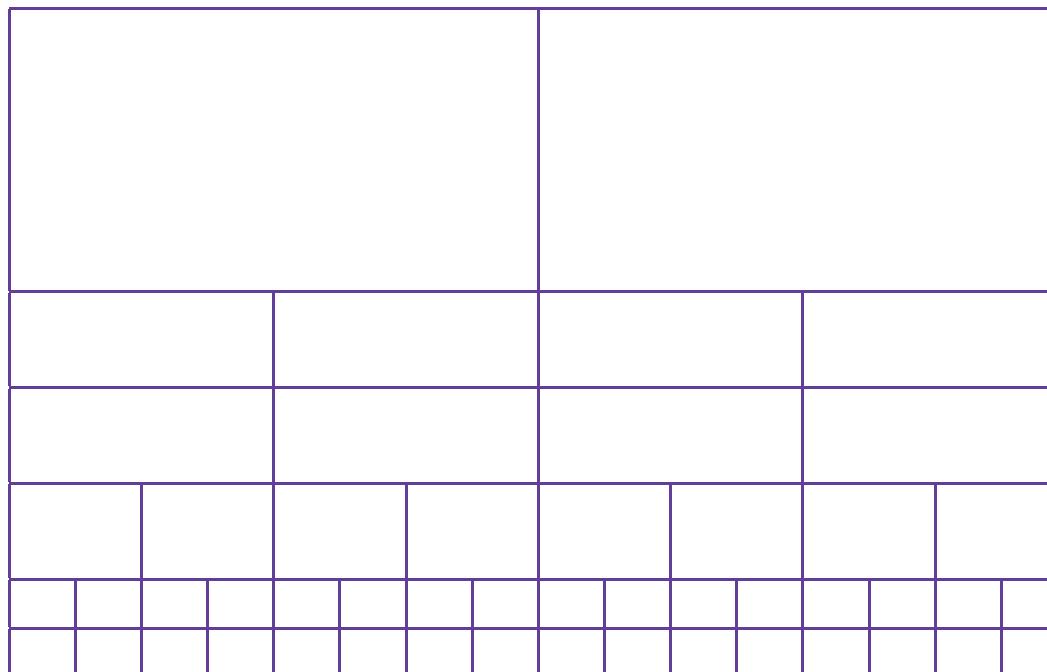
- 2D:

standard: $w = 4$; adjusted: $w = 4J_0(\kappa h) + (\kappa h)^2 = 4 + O(h^4)$

- 3D:

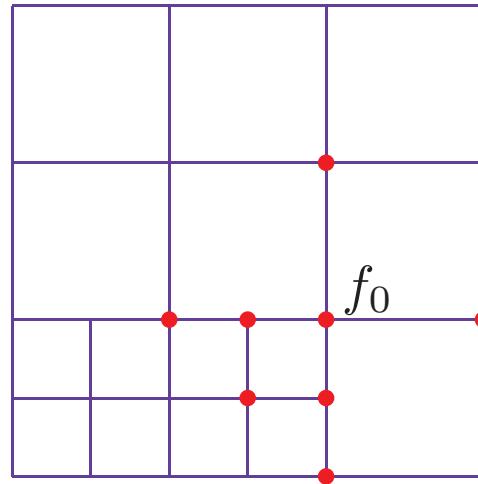
standard: $w = 6$; adjusted: $w = 6j_0(\kappa h) + (\kappa h)^2 = 6 + O(h^4)$

Important Issue: Interpolation



- highly desirable to subdivide only certain parts of a given mesh
- want precise optimal weights where a refinement intersects the original mesh

There are three cases: middle node, cross node, and corner node interpolation. Consider corner node only today:



Given the plane wave $f = e^{\kappa(x \cos(\phi) + y \sin(\phi))j}$ the nodes are labelled counterclockwise and so one has values $f_0 = f(0, 0) = 1$, $f_1 = f(2h, 0)$, $f_2 = f(0, -h)$, $f_3 = f(0, -2h)$, $f_4 = f(-h, -h)$, $f_5 = f(-h, 0)$, $f_6 = f(-2h, 0)$, and $f_7 = f(0, 2h)$.

- Approximate the field f_0 in the center of the stencil as a function of the values at neighboring nodes:

$$(*) \quad \sum_{i=0}^7 \frac{w_i f_i}{h^2} = \text{Res}(\phi) \approx 0.$$

- Assume all angles are equally probable, and exploit symmetry. So solve $\int_0^{2\pi} h^2 \text{Res}(\phi) d\phi = 0$ for w_0 (to get w_0 as a function of w_1, \dots, w_7 and Bessel functions in κh).

- plug this into $(*)$ to obtain an equation

$W_1 F_1 + W_2 F_2 + W_3 F_3 = R$ for the other weights (where $W_1 = w_2$, $W_2 = w_3$, and $W_3 = w_4$).

Specifically, one has

$$\mathbf{F}_1 = e^{-\kappa h \sin(\phi)} \mathbf{j} + e^{-\kappa h \cos(\phi)} \mathbf{j} - 2J_0(\kappa h)$$

$$\mathbf{F}_2 = e^{-2\kappa h \sin(\phi)} \mathbf{j} + e^{-2\kappa h \cos(\phi)} \mathbf{j} - 2J_0(2\kappa h)$$

$$\mathbf{F}_3 = e^{-\kappa h (\sin(\phi) + \cos(\phi))} \mathbf{j} - J_0(\sqrt{2}\kappa h)$$

$$\mathbf{R} = 2J_0(2\kappa h) - e^{2\kappa h \cos(\phi)} \mathbf{j} - e^{2\kappa h \sin(\phi)} \mathbf{j}$$

and the problem is to solve (i.e. optimally approximate)

$$W_1 \mathbf{F}_1 + W_2 \mathbf{F}_2 + W_3 \mathbf{F}_3 = \mathbf{R}$$

for the W_i . One has analogous problems for each stencil case above.

Generally, given functions f_1, \dots, f_n, r as above, one can best approximate

$w_1 f_1 + \dots w_n f_n = r$ as follows:

- Use Gram-Schmidt to construct

$$p_1 = f_1$$

$$p_2 = f_2 - \frac{\langle f_2, p_1 \rangle}{\langle p_1, p_1 \rangle} p_1$$

$$p_3 = f_3 - \frac{\langle f_3, p_1 \rangle}{\langle p_1, p_1 \rangle} p_1 - \frac{\langle f_3, p_2 \rangle}{\langle p_2, p_2 \rangle} p_2$$

⋮

where

$$\langle f, g \rangle = \int_0^{2\pi} f(\phi) \overline{g(\phi)} d\phi$$

One has

$$\tilde{r} = \frac{\langle r, p_1 \rangle}{\langle p_1, p_1 \rangle} p_1 + \frac{\langle r, p_2 \rangle}{\langle p_2, p_2 \rangle} p_2 + \frac{\langle r, p_3 \rangle}{\langle p_3, p_3 \rangle} p_3 + \dots$$

and this can be expressed as a linear combination of the original f_i . But trying this “by hand” is

- tedious
- error prone

So, one would like to use symbolic computation on a computer, but

- the algebra is quite complex
- the usual systems can't do the integrals automatically

The most familiar symbolic computational systems, Mathematica, Maple, REDUCE, MACSYMA, Matlab, AXIOM, MuPad, ..., all have some common features:

- introduce a new programming language
- tied to some underlying language or kernel
- interpreted
- allow for convenient manipulation of mathematical expressions
- have built in the usual mathematical operations

A new approach is taken in an ANSI C library called ExprLib . It differs from all the other systems mentioned in some important ways.

ExprLib

- an optimized ANSI C library
- mathematical structures and operations are based on their mathematical foundations – no pattern matching or rewrite rules used for basic operations
- it is a kind of *algebra of compiled functions*
- default is floating point (double) coefficients – bignums an option
- allows user defined operators, derivatives, integrals
- allows sophisticated user defined rewrite routines

Anatomy of a Mathematical Expression

Consider

$$f = \frac{x \sin(ax^2 - 6x + y) - x^3y + 6}{a \cos(3xy + 1) - 6}$$

- there is a numerator and a denominator
- numerator and a denominator are both polynomials in expressions

Parameters

- a variable, a symbolic constant or the application of an operator to an expression
- parameters above are $x, \sin(ax^2 - 6x + y), a, y, \cos(3xy + 1)$

Main Types

- Expr, Op, String

Basic Operations

- Expr `parseStringToExpr (String s);`
- String `exprToStr (FmtStr fmt, Expr f);`
- `exprPlus, exprMinus, exprTimes, exprPow, exprDiv, exprDiff`
- `exprNEval, exprEval`

More Complex Operations

- `exprNum, exprDen, exprSummands, exprFactors, exprIsOpApp,`
`exprParSub`

- using the decomposition functions, one can easily obtain the “parse tree” of any expression
- using the substitution functions, one can transform any expression in very flexible ways
- using combinations of these, specialized “rewrite rule” transformations can easily be programmed.
- for example, using this, one can perform symbolic integration for large classes of examples, e.g. consider

$$\int_0^{2\pi} e^{x \cos(\phi+a)} j d\phi = 2\pi J_0(x)$$

**n expressions in n variables
(n <= 5):**

CGI/HTML & ExprLib

```
a**2*u*x + 3*a*u*x + 1;  
-a**2*y**2 + 2*a*u*y;  
-u**2*x*y + a**2*u*x;  
-u + a**3;
```

initial values:

```
a=1.0; u=1.0; x=-0.5; y=1.0;
```

xtol:

```
1.0E-10    maxfev: 10000
```

factor:

```
10.0
```

Submit Query

Reset

```
c Expr to FORTRAN translator result
    call ftndriver
    end
c
subroutine ftndriver
integer n,ldfjac,maxfev,mode,nprint,info,nfev,njev,lr
double precision xtol,factor
double precision x(4),fvec(4),fjac(4,4),diag(4),r(6),
* qtf(4),wa1(4),wa2(4),wa3(4),wa4(4)
external fcnrl
n=4
c a=x(1) u=x(2) x=x(3) y=x(4)
x(1)=1.0d0
x(2)=1.0d0
x(3)=-0.5d-01
x(4)=1.d0
ldfjac=4
xtol=1.0d-11
maxfev=10000
mode=1
factor=1.0d+01
nprint=0
call hybrj(fcnrl,n,x,fvec,fjac,ldfjac,xtol,maxfev,diag,mode,
*factor,nprint,info,nfev,njev,r,lr,qtf,wa1,wa2,wa3,wa4)
iflag=1
```

```

call fcnrl(n,x,fvec,fjac,ldfjac,iflag)
write(6,9)info,x(1),x(2),x(3),x(4),
*fvec(1),fvec(2),fvec(3),fvec(4),nfev,njev
9   format('info=',i1,/,
*'x(1)=', d15.8, '/',
*'x(2)=', d15.8, '/',
*'x(3)=', d15.8, '/',
*'x(4)=', d15.8, '/',
*'f1(x)=', d15.8, '/',
*'f2(x)=', d15.8, '/',
*'f3(x)=', d15.8, '/',
*'f4(x)=', d15.8, '/',
*'#calls to f:',i4, '#calls to df:',i2, /)
      return
    end
c
subroutine fcnrl(n,x,fvec,fjac,ldfjac,iflag)
integer n,ldfjac,iflag
double precision PI
double precision x(n),fvec(n),fjac(ldfjac,n)
PI=4.0d0*atan(1.0d0)
if(iflag.eq.1)then
  fvec(1)=(x(1)**2 + 3.0d0 * x(1)) * x(2) * x(3) + 1.0d0
  fvec(2)=- x(1)**2 * x(4)**2 + 2.0d0 * x(1) * x(2) * x(4)
  fvec(3)=- x(2)**2 * x(3) * x(4) + x(1)**2 * x(2) * x(3)
  fvec(4)=- x(2) + x(1)**3

```

- built in operators: `sin, cos, tan, sinh, ...`
- create an `n`-ary operator with “print name” `myOp`

```
myOp = opCreate ("myOp", n);
```

- register the new operator in the parser table

```
registerOp (myOp);
```

- associate a compiled function to the new operator

```
addNumFun (myOp, myOpFun);
```

- other kinds of functions may be “attached”

- numerically evaluating expressions – faster than: mathematica: 40-50 times, Maple: 30-40 times

So, given a problem like: solve the equation

$$W_1 F_1 + W_2 F_2 + W_3 F_3 = R$$

for W_i when F_i and R are given, the strategy is to

- define a binary operator “brack” and register it with the parser
- “teach” the library that brack is complex bilinear:

$$\text{brack}(ax, y) = a \text{brack}(x, y),$$

$$\text{brack}(y, x) = \overline{\text{brack}(x, y)},$$

$$\text{brack}(x + y, z) = \text{brack}(x, z) + \text{brack}(y, z),$$

etc.

- teach the library how to integrate the relevant class of functions

With this done, the main program looks like

- program the G-S formula and the formula ans given by

$$\frac{\langle s, p_1 \rangle}{\langle p_1, p_1 \rangle} p_1 + \frac{\langle s, p_2 \rangle}{\langle p_2, p_2 \rangle} p_2 + \frac{\langle s, p_3 \rangle}{\langle p_3, p_3 \rangle} p_3$$

symbolically in terms of symbolic expressions f_i representing the F_i (by bilinearity, the formula will automatically be expressed as a function of $\text{brack}(f_i, f_j)$, $\text{brack}(r, f_j)$ and will be linear in the f_i)

- pick out the coefficients, e.g. the coefficient of f_1 is just $\text{exprDiff}(\text{ans}, f_1)$
- calculate the brackets for the actual F_1, F_2, F_3, R
- use substitutions to replace the symbolic brackets in the coefficients by their actual values computed above

For example, we find that

$$\begin{aligned}
 W_3 = & ((- 2 J_{s2} + 2 J_{s10}) J_{s5}^2 + ((2 J_0 J_4 + (2 J_2 - 2) \\
 & J_3 + 6 J_0 J_2 - 4 J_0) J_{s2} + (- 8 J_0 J_2 + 4 J_0) J_{s10} - 2 \\
 & J_0 J_4 + 2 J_0) J_{s5} + ((- 2 J_2 + 2) J_4 + 2 J_{2s2} - 4 J_2^2 + \\
 & 2 J_2) J_{s2}^2 + ((- 2 J_{2s2} + 4 J_2^2 - 2) J_{s10} + (- 2 J_2 - \\
 & 2 J_0^2 + 2) J_4 + (- 2 J_0 J_{2s2} + 6 J_0 J_2 - 4 J_0) J_3 + \\
 & (- 2 J_0^2 + 2) J_{2s2} - 4 J_2^2 + (2 J_0^2 + 2) J_2) J_{s2} + ((4 \\
 & J_0^2 - 2) J_{2s2} + 4 J_2^2 - 8 J_0^2 J_2 + 6 J_0^2 - 2) J_{s10} + \\
 & (4 J_0^2 J_2 - 2 J_0^2) J_4 + (2 J_0 J_{2s2} - 4 J_0 J_2^2 + \\
 & 2 J_0) J_3 - 2 J_0^2 J_{2s2} + 4 J_0^2 J_2^2 - 4 J_0^2 J_2) / \\
 & ((J_{s2}^2 - 1) J_{s5}^2 + (- 2 J_0 J_{s2}^2 + (- 4 J_0 J_2 + 4 J_0) \\
 & J_{s2} + 4 J_0 J_2 - 2 J_0) J_{s5} + (- J_{2s2} + 4 J_2 - 3) J_{s2}^3 + \\
 & (- J_{2s2} + 4 J_2 + J_0^2 - 3) J_{s2}^2 + ((4 J_0^2 + 1) J_{2s2} - 2 \\
 & J_2^2 - 12 J_0^2 J_2 + 8 J_0^2 + 1) J_{s2} + (- 4 J_0^2 + 1) J_{2s2} + \\
 & (4 J_0^2 - 2) J_2^2 + 4 J_0^2 J_2 - 5 J_0^2 + 1)
 \end{aligned}$$

where $J_0 = J_0(\kappa h)$, $J_2 = J_0(2\kappa h)$, $J_3 = J_0(3\kappa h)$, $J_4 = J_0(4\kappa h)$, $J_{s2} = J_0(\sqrt{2}\kappa h)$, $J_{s5} = J_0(\sqrt{5}\kappa h)$, $J_{s10} = J_0(\sqrt{10}\kappa h)$, and $J_{2s2} = J_0(2\sqrt{2}\kappa h)$

```
Expr  
obrack (Expr f, Expr g, String *s, UInt n)  
{ Expr ans = exprZero ();  
  Expr coi, coj, base;  
  int i, j;  
  
  if (exprIsZero (f) || exprIsZero (g))  
    return exprZero ();  
  else  
  {  
    for (i = 1; i < n; i++)  
      for (j = 1; j < n; j++)  
        { coi = exprDiff (f, s[i]);  
          coj = exprDiff (g, s[j]);  
          coi = exprTimes (coi, obar (coj));  
          base =  
            opApp2 (brack, parseStrToExpr (s[i]),  
                    parseStrToExpr (s[j]));  
          ans = exprPlus (ans, exprTimes (coi, base));  
        }  
  }  
  return ans;  
}
```

```
void
rules (Expr lhs[], Expr rhs[])
{
    /* do complex conjugation */ 
    lhs[0] = parseStrToExpr ("sin (fi)");
    rhs[0] = parseStrToExpr ("- sin (fi)");
    lhs[1] = parseStrToExpr ("cos (fi)");
    rhs[1] = parseStrToExpr ("- cos (fi)");
    /* apply trig ids */ 
    lhs[2] = parseStrToExpr ("j * kh * (sin (fi) - cos (fi))");
    rhs[2] = parseStrToExpr ("j * kh * rt2 * cos(fi)");
    .
    .
    .
    /* do the integration */ 
    lhs[7] = parseStrToExpr ("exp (j * kh * rt2 * cos (fi))");
    rhs[7] = parseStrToExpr ("Js2");
    lhs[8] = parseStrToExpr ("exp (j * kh * cos (fi))");
    rhs[8] = parseStrToExpr ("J0");
    .
    .
    .
}
```

```
Expr  
brackF1F1 (Expr e)  
{  
    Expr p;  
    p = e;  
    p = exprParSubDeep (p, lhs[0], rhs[0]);  
    p = exprParSubDeep (p, lhs[1], rhs[1]);  
    p = exprTimes (e, p);  
    p = exprFixExps (p);  
    p = exprPolSubDeep (p, lhs[2], rhs[2]);  
    p = exprPolSubDeep (p, lhs[3], rhs[3]);  
    p = exprPolSubDeep (p, lhs[4], rhs[4]);  
    p = exprPolSubDeep (p, lhs[5], rhs[5]);  
    p = exprPolSubDeep (p, lhs[6], rhs[6]);  
    p = exprParSubDeep (p, lhs[7], rhs[7]);  
    p = exprParSubDeep (p, lhs[8], rhs[8]);  
    p = exprTimes (p, parseStrToExpr ("2 * PI"));  
    return (p);  
}
```

For more information, copies of the source code, etc., contact

Larry Lambe

MSSRC, P.O. Box 6667, Bloomindale, IL 60108

llambe@mssrc.com

or

Richard Luczak

ASC/HP, Bldg. 676 – Room 112, 2435 5th Street, WPAFB, OH 45433-7802

luczakr@asc.hpc.mil

or

John Nehrbass

ASC/HP, Bldg. 676 – Room 112, 2435 5th Street, WPAFB, OH 45433-7802

nehrbass@osc.edu

For interactive demos of ExprLib, visit

<http://www.mssrc.com/demos.html>